Lesson 7 Plotting with Seaborn Last update Sept 12, 2020

Seaborn offers much more plotting capabilities then **pyplot**. Seaborn actually uses pyplot as a base, but provides much more enhancing plotting capabilities.

To use pyplot and seaborn you import them as follows:

import matplotlib.pyplot as plt
import seaborn as sns

You need to have seaborn version 9.0 installed
conda install -c anaconda seaborn=0.9.0
`pip install seaborn==0.9.0`

Types of Plots available with Seaborn

relation plots

Relation plots makes use of two other axes functions for Visualizing Statistical Relationships which are:

- scatterplots
- lineplots

lineplots

A line plot is used to plot lines between 2 variables. As an example we will use a data frame filled with the amount of money spent per day for groceries as our data.

data=pd.DataFrame({'Day':[1,2,3,4,5,6,7],'Grocery':[30,80,45,23,51,46,76]})

print(data)

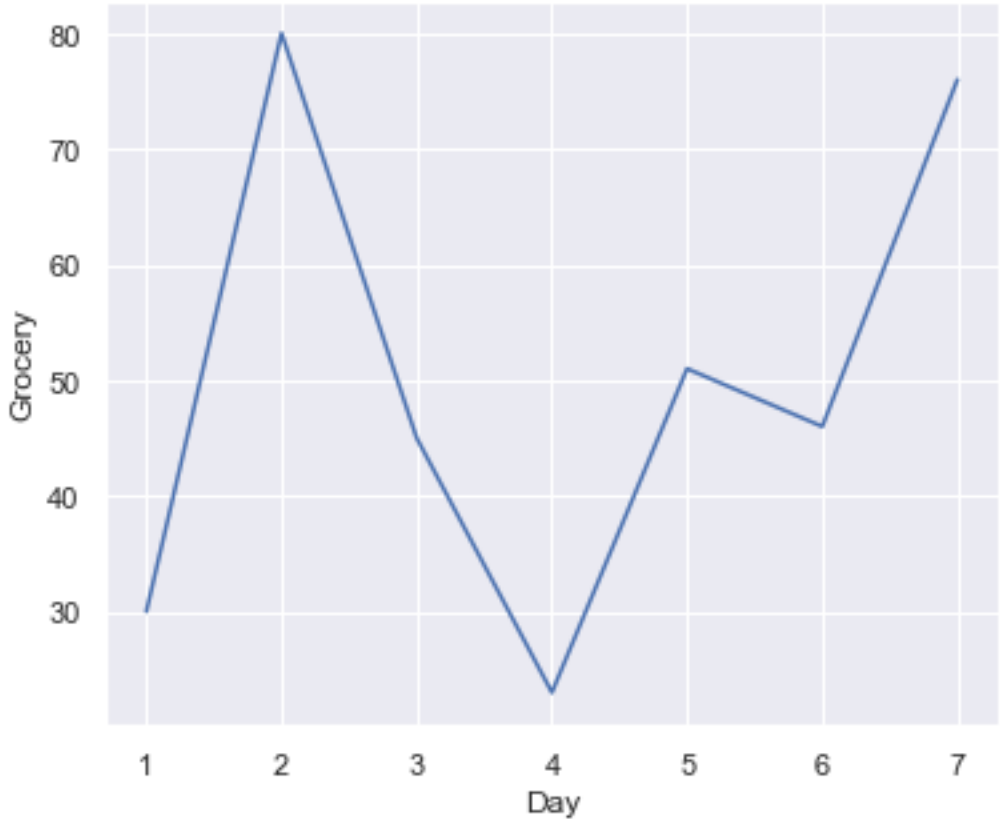
```
   Day  Grocery
0    1       30
1    2       80
2    3       45
3    4       23
4    5       51
5    6       46
6    7       76
```

We use pyplot **plt.figure** function to set our plot size to 6 by 5 inches

plt.figure(figsize=(6,5))

We then plot the data with the seaborn lineplot function

sns.lineplot(x="Day", y="Grocery", data=data)



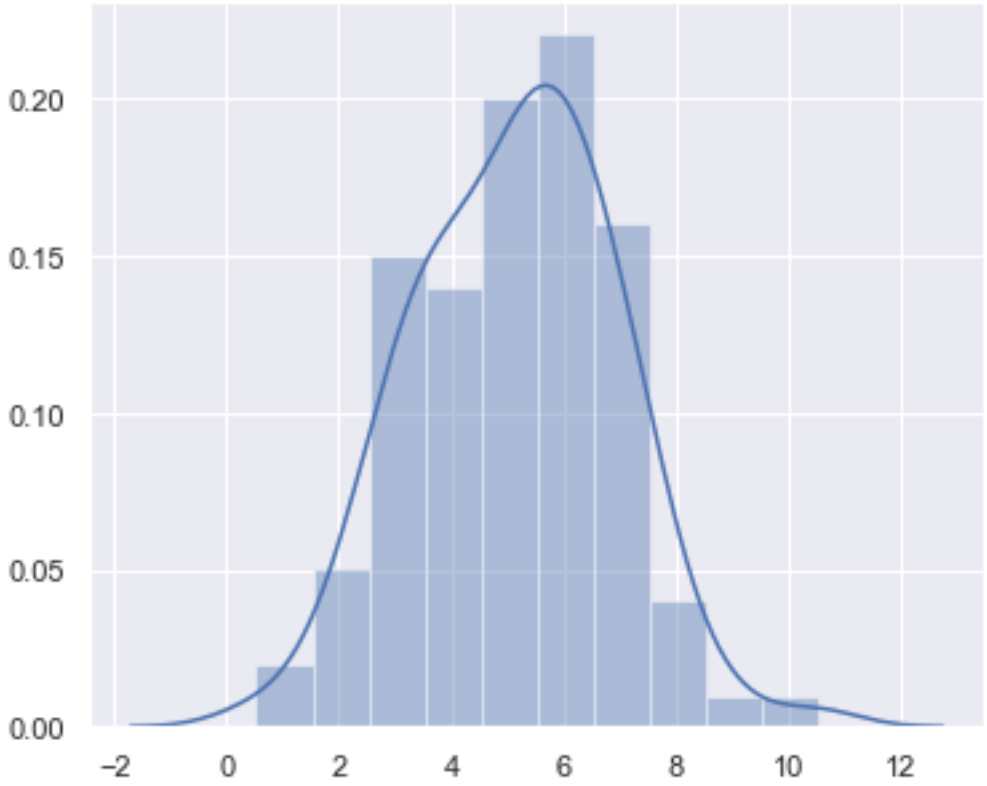
Plotting Univariate distributions:

Univariate means "one variable" (one type of data like temperature). If you have two sets of data, such as population sales vs area, then it is called "**Bivariate Data"**

To plot **Univariate distributions** you use the **distplot()** function as follows:

We can plot the normal distribution curve of 100 points centered at 5 with deviation of 2 as follows:

```
plt.figure(figsize=(6,5))
data = np.random.normal(loc=5,size=100,scale=2)
sns.distplot(data);
```



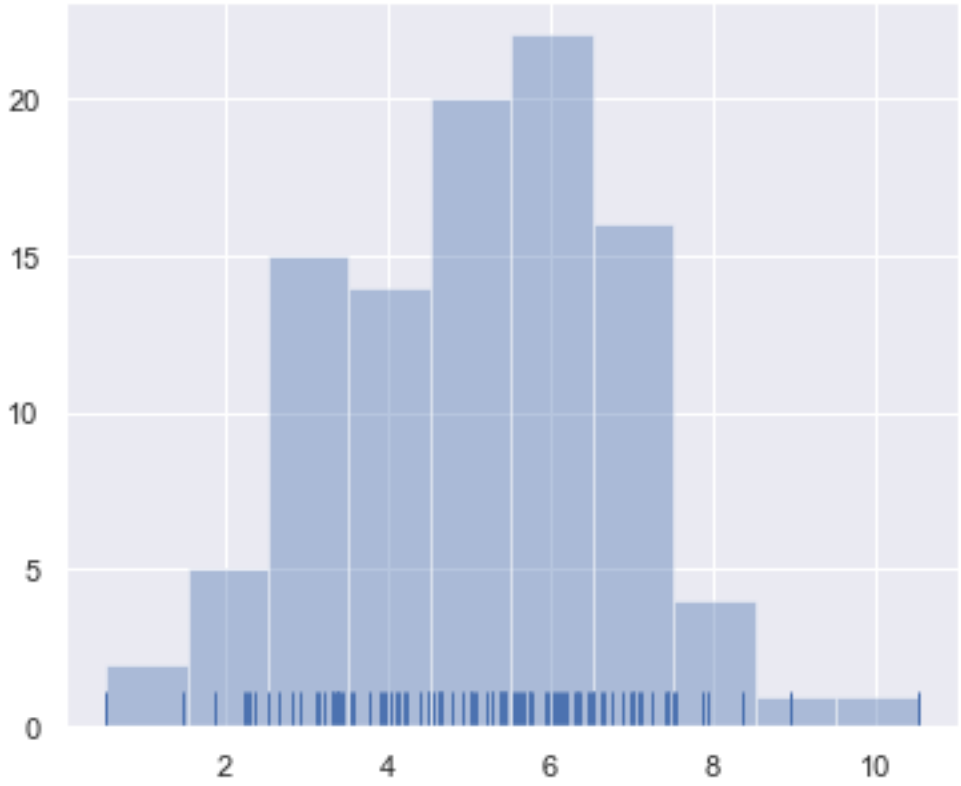
We have plotted a graph using **distplot** for normal distribution curve of 100 points centered at 5 with deviation of 2.

Plotting histogram

A histogram represents the distribution of data by forming bins along the range of the data and then bars are drawn to show the number of observations that fall within in each bin.

We have removed the default kernel density estimation (kde) curve and added an additional rug plot instead, which draws a small vertical tick for each observation at the bottom of the plot.

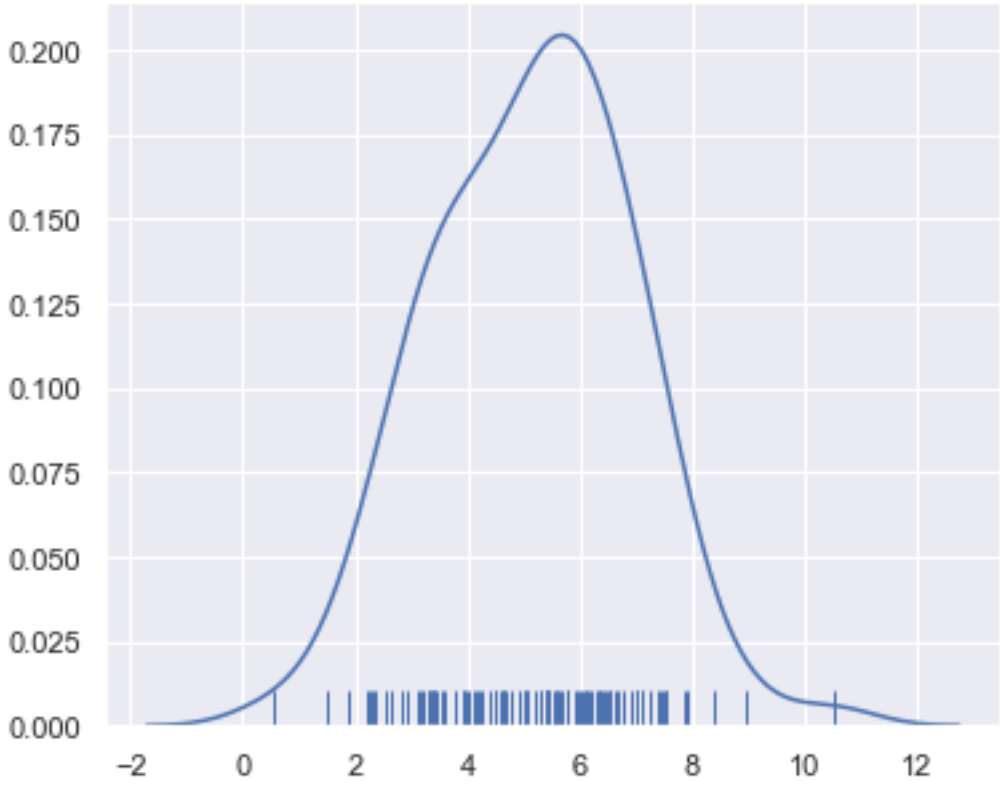
```
plt.figure(figsize=(6,5))
sns.distplot(data, kde=False, rug=True);
```



Plotting **Kernel density estimation KDE**

The **kernel density estimate KDE** is used to plotting the shape of a distribution. The KDE plot encodes the density of observations on one axis with height along the other axis:

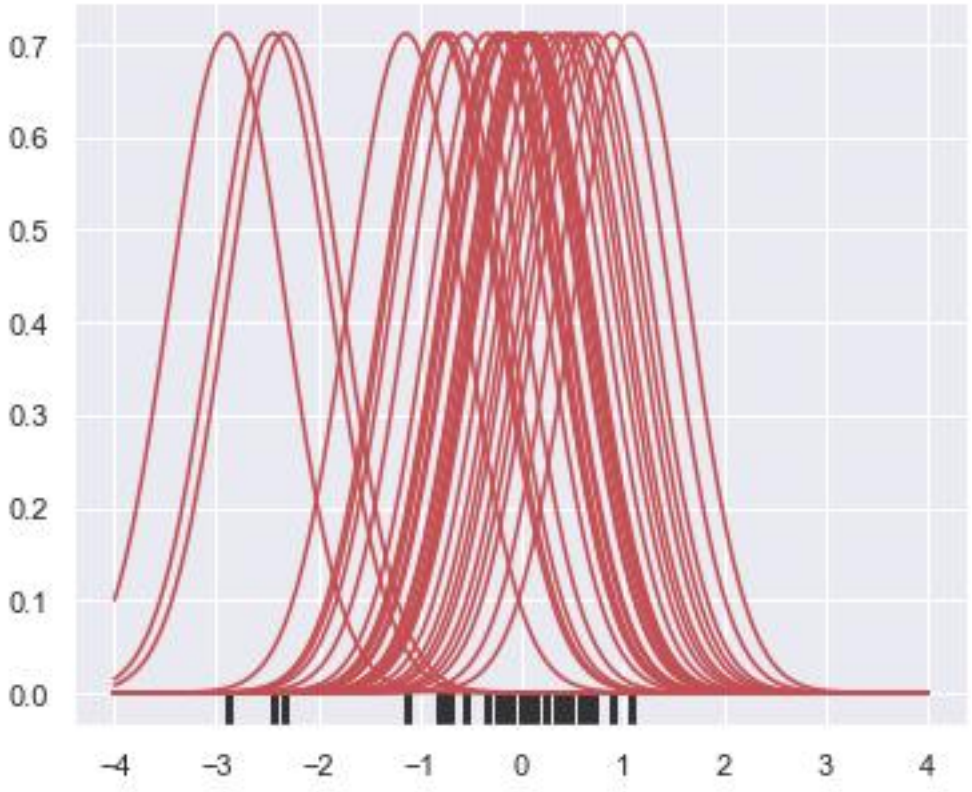
```
plt.figure(figsize=(6,5))
sns.distplot(data, hist=False, rug=True);
```



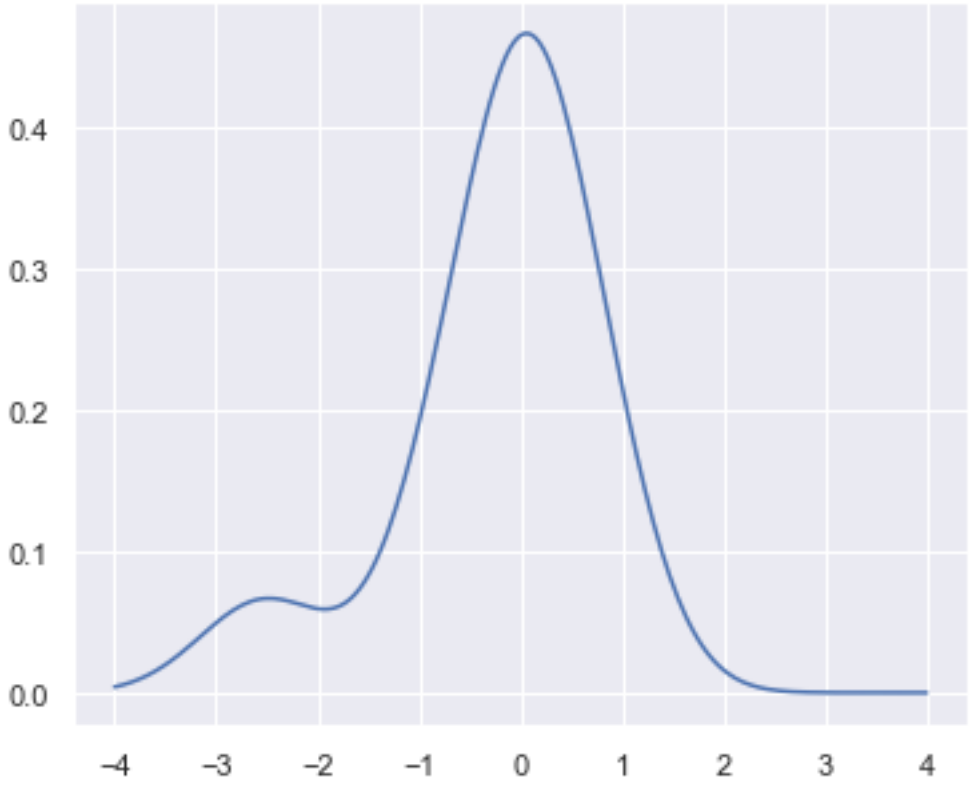
Drawing a KDE

In drawing a KDE each observation is first replaced with a normal (Gaussian) curve centered at that value:

```
plt.figure(figsize=(6,5))
x = np.random.normal(0, 1, size=30)
bandwidth = 1.06 * x.std() * x.size ** (-1 / 5.)
support = np.linspace(-4, 4, 200)
kernels = []

for x in data:

    kernel = scipy.stats.norm(x, bandwidth).pdf(support)
    kernels.append(kernel)
    plt.plot(support, kernel, color="r")
    sns.rugplot(data, color=".2", linewidth=3)
```

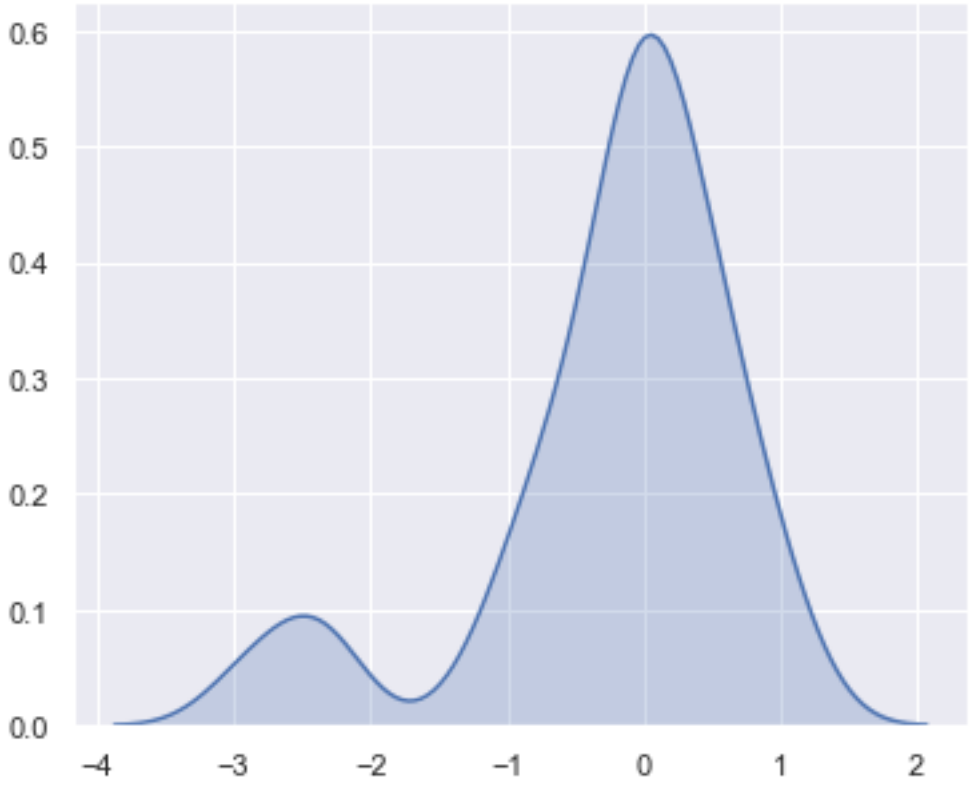


Next, these curves are summed to compute the value of the density at each point in the support grid. The resulting curve is then normalized so that the area under it is equal to 1:



The **`kdeplot`** function in seaborn, results in the same curve

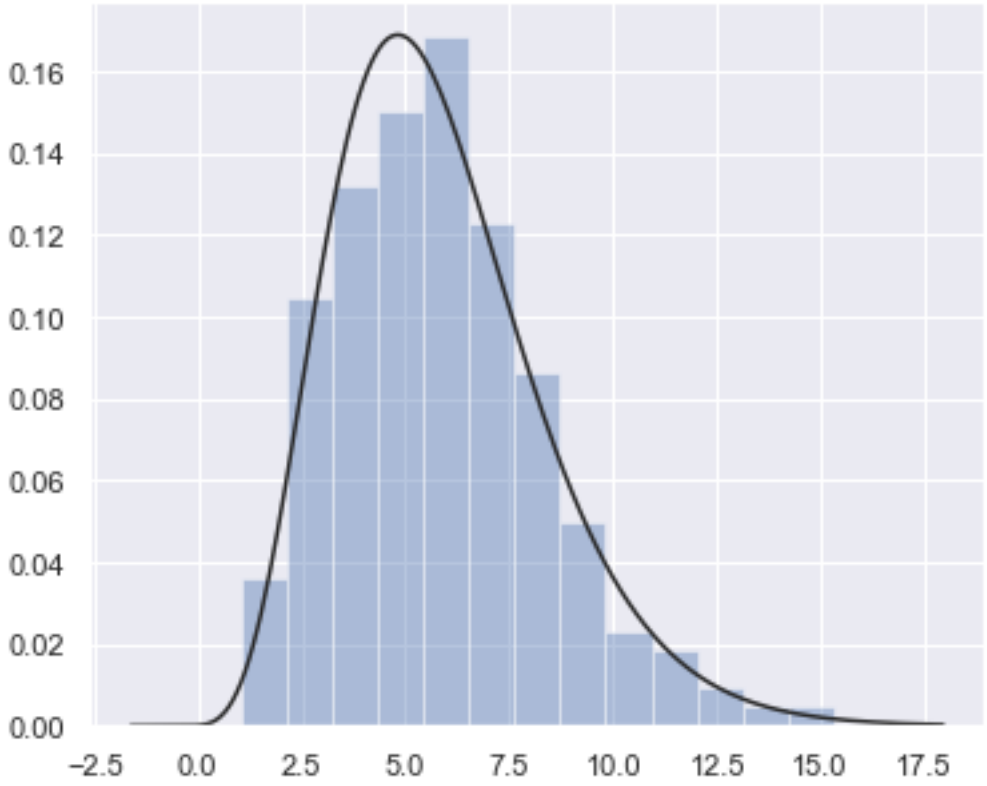
```
plt.figure(figsize=(6,5))
sns.kdeplot(data, shade=True);
```



Fitting parametric distributions

A **distplot** is used to fit a parametric distribution to a dataset and visually evaluate how closely it corresponds to the observed data. A parametric distribution is used in statistics when an assumption is made of the way the underlying data is distributed. We use the **scipy stats.gamma** function for fitting.

```
plt.figure(figsize=(6,5))
data = np.random.gamma(6, size=200)
sns.distplot(data, kde=False, fit=scipy.stats.gamma);
```



Plotting bivariate distributions using Scatter plot

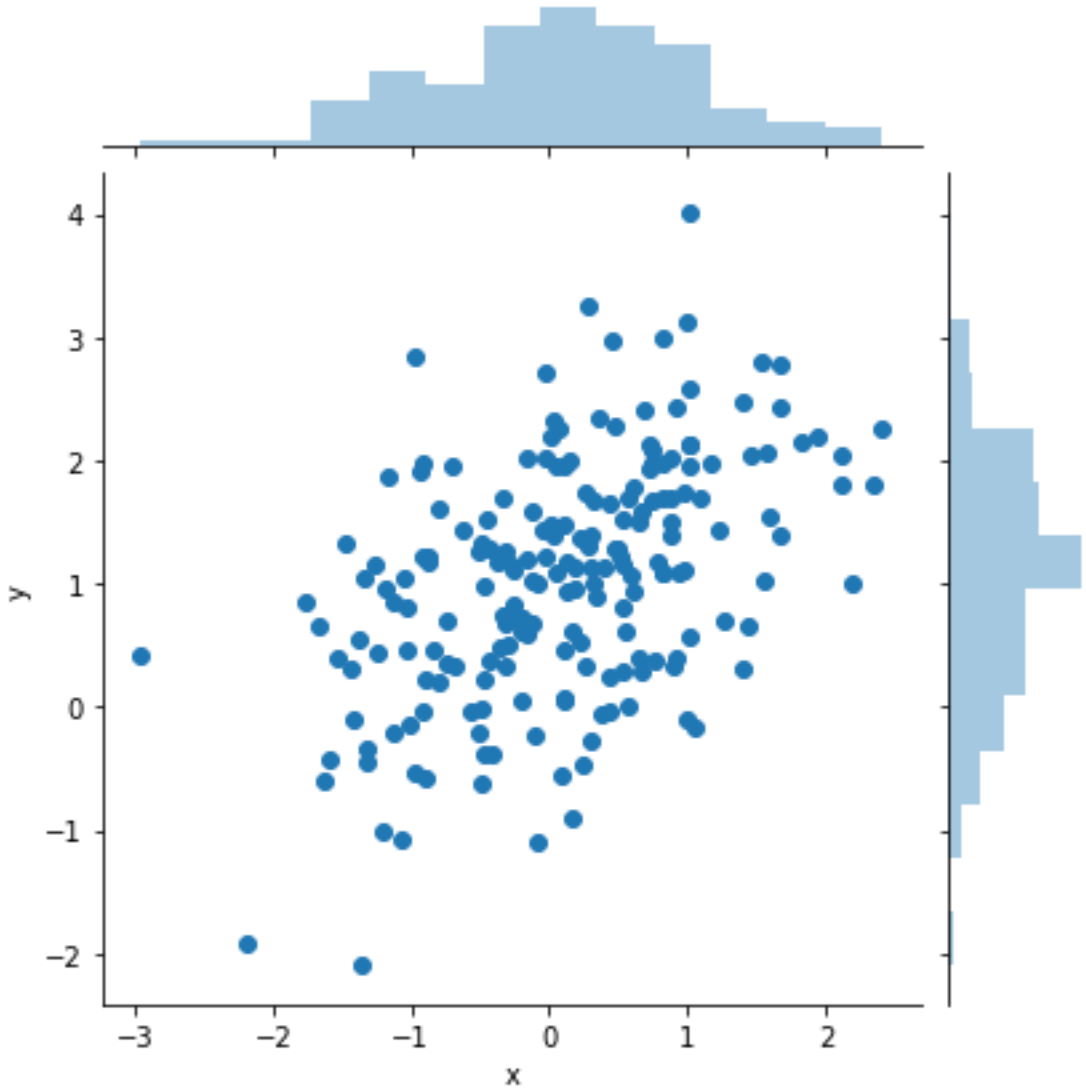
Bi variate data uses two variables like population and area. The **jointplot()** funcrtion is used for plotting **bivariate distributions.**

The **jointplot** function, creates a multi-panel figure that shows both the bivariate (or joint) relationship between two variables along with the univariate (or marginal) distribution of each on separate axes. **Correlation** is a measure used to represent how strongly two random variables are related known . The value of **correlation** takes place between -1 and +1. **Covariance** is nothing but a measure of **correlation**. Correlation is when the change in one item may result in the change in another item. Covariance is when two items vary together

The multivariate normal, multi-normal or Gaussian distribution is a generalization of the one-dimensional normal distribution to higher dimensions. Such a distribution is specified by its mean and covariance matrix. These parameters are analogous to the mean (average or “center”) and variance (standard deviation, or “width,” squared) of the one-dimensional normal distribution.

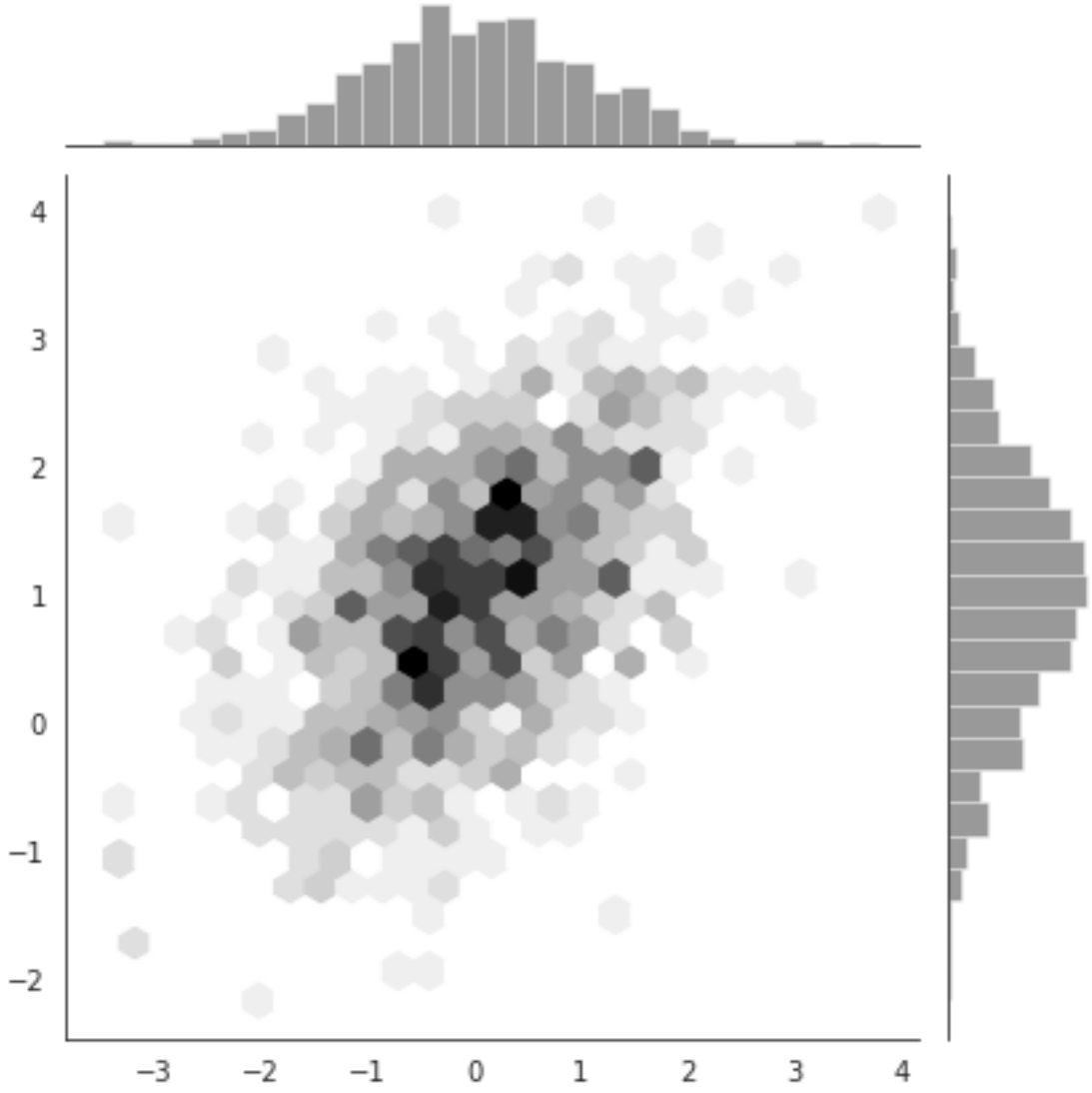
The scatter plot is the default.

```
plt.figure(figsize=(6,5))
mean = [0, 1]
cov =  [(1, .5), (.5, 1)]
data = np.random.multivariate_normal(mean, cov, 200)
df = pd.DataFrame(data, columns=["x", "y"])
sns.jointplot(x="x", y="y", data=df);
```



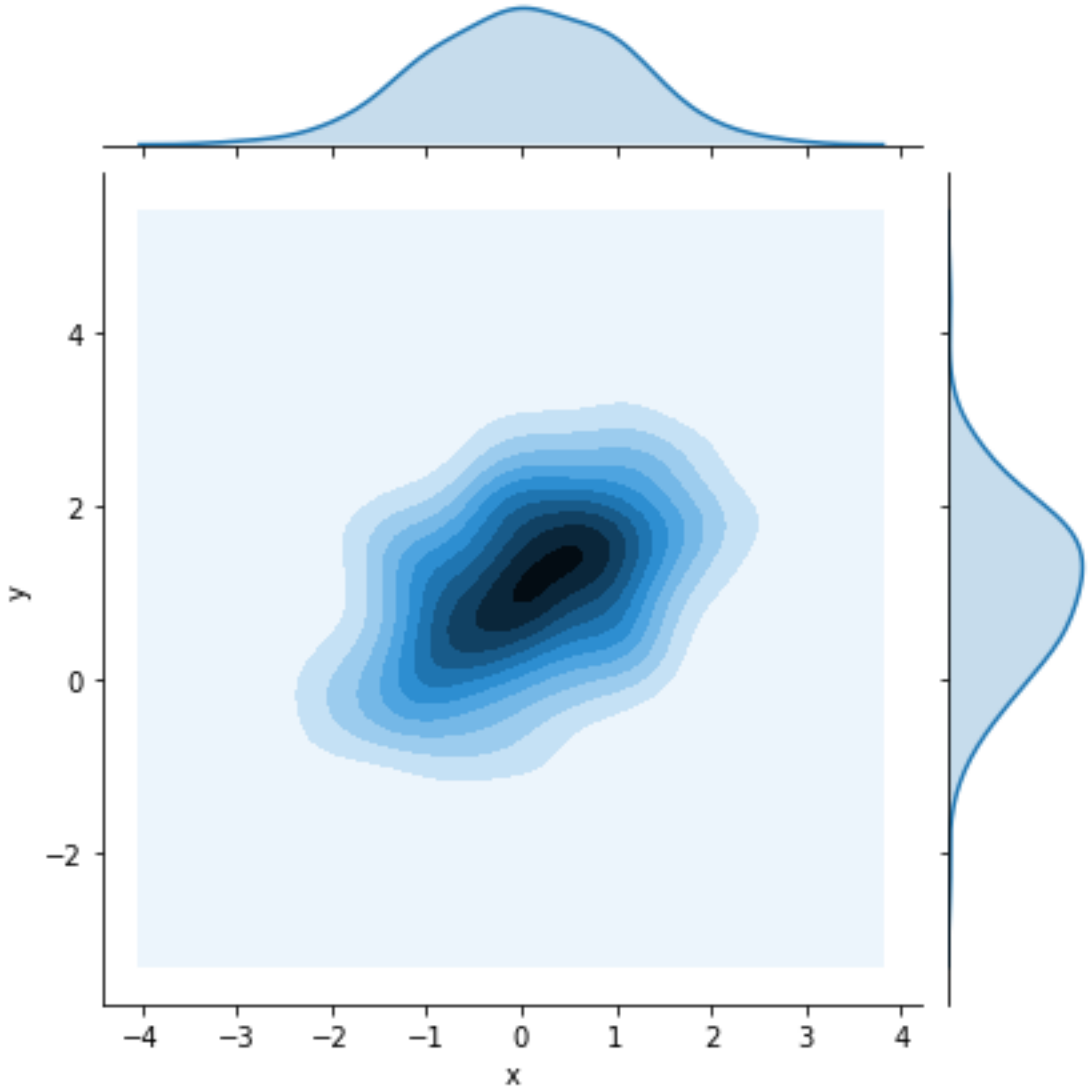
Hexbin plots

A **"hexbin"** plot is the bivariate analogue of a histogram. It shows the counts of observations that fall within hexagonal bins. This plot works best with relatively large datasets

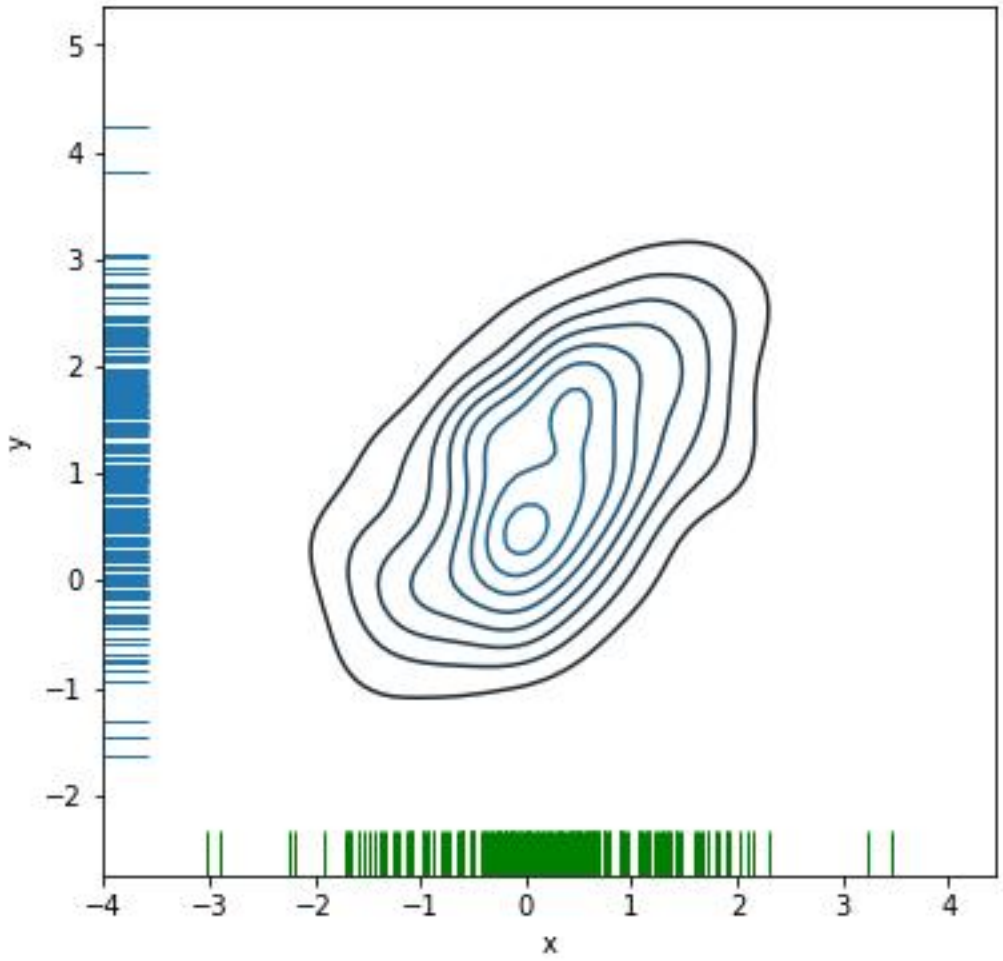


Using Kernel density estimation

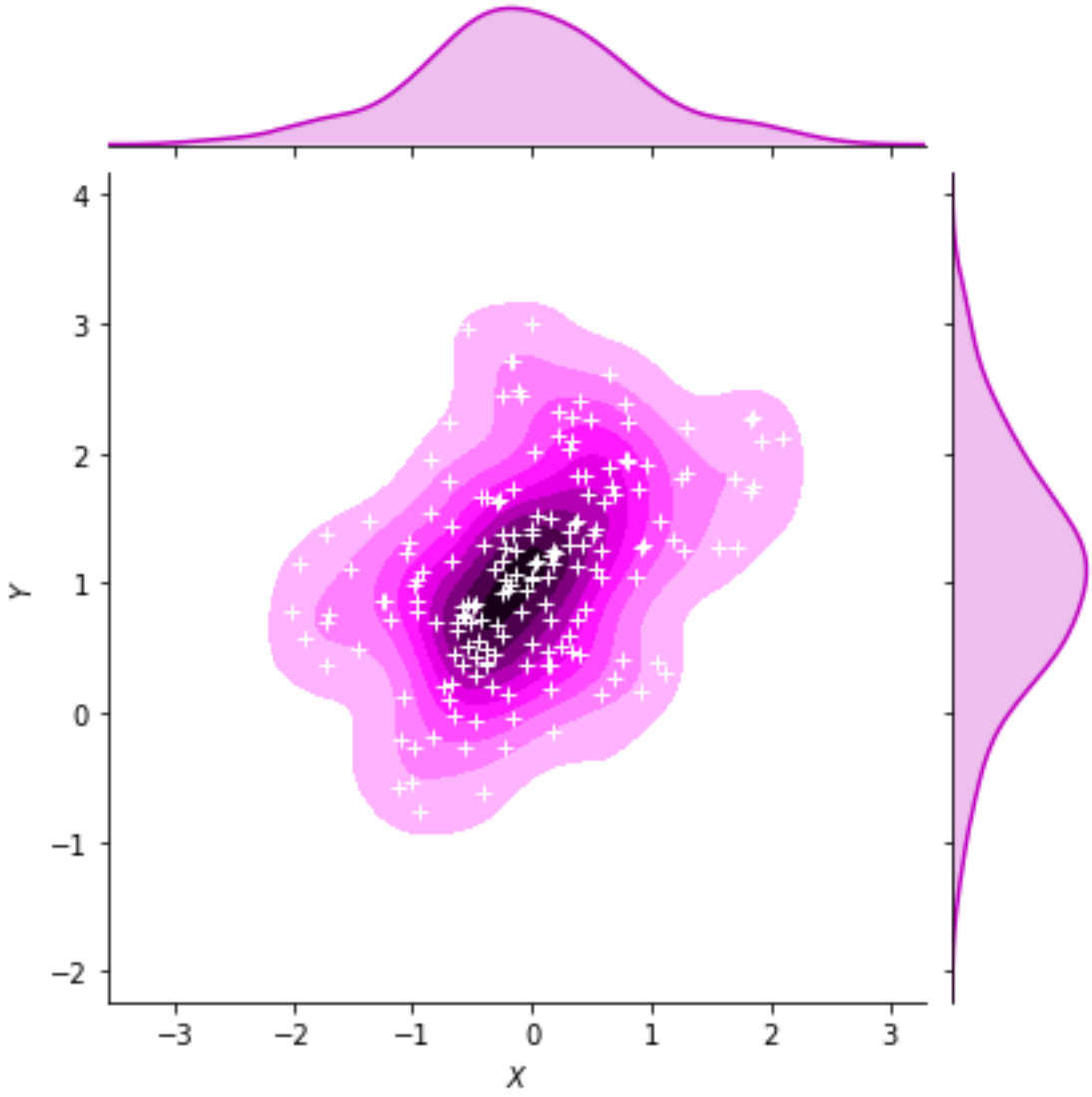
It is also possible to use the kernel density estimation procedure described above to visualize a bivariate distribution. This kind of plot is shown with a contour plot .



You can also draw a two-dimensional kernel density plot with the **kdeplot** function. This allows you to draw this kind of plot onto a specific (and possibly already existing) matplotlib axes, whereas the **jointplot** function manages its own figure:

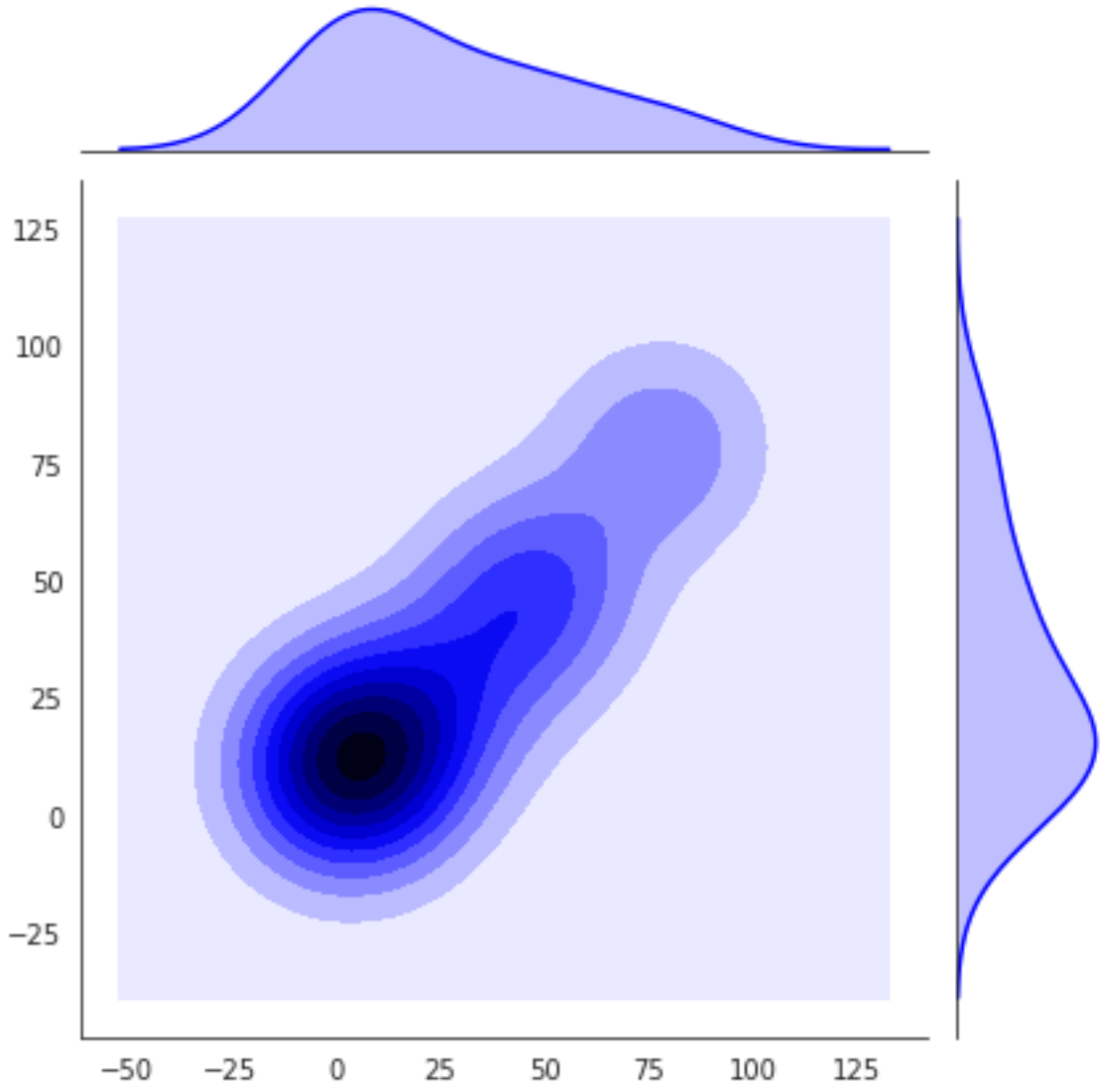


The **jointplot** function uses a **JointGrid** to manage the figure. For more flexibility, you may want to draw your figure by using a **JointGrid** directly. The **jointplot** function returns the ****JointGrid** object after plotting, which you can use to add more layers or to tweak other aspects of the visualization:



Here we plot the amount paid for groceries each day.

```
x=pd.DataFrame({'Day':[1,2,3,4,5,6,7],'Grocery':[30,80,45,23,51,46,76]})
y=pd.DataFrame({'Day':[8,9,10,11,12,13,14],'Grocery':[30,80,45,23,51,46,76]})
mean, cov = [0, 1], [(1, .5), (.5, 1)]
data = np.random.multivariate_normal(mean, cov, 200)

with sns.axes_style("white"):

        sns.jointplot(x=x, y=y, kind="kde", color="b");
```



The plot resembles a contour map with the densities in the top and right side.

Pair Plot

Pair Plots are used to visualize relationships between each variable in your data. A pair plot produces a matrix of relationships between each variable in your data for an instant visualization of your data. A Pair Plot will take each numerical column of a data frame and put them on both the x and y axes and plot as a scatter plot where they meet. In cases where the same column variables meet, a histogram is drawn that shows the distribution of the variables. An optional regression line may be drawn on the scatter plot.

We will use the Grocery data from above for our data. We will also add a third column for the percent of money spent each day.

```
data=pd.DataFrame(
 {'Day':['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday'],
      'Grocery':[30,80,45,23,51,46,76],
      'Percent':[.2,.4,.5,.3,.2,.3,.6]})
print(data)
```

```
         Day  Grocery  Percent
0     Monday       30      0.2
1    Tuesday       80      0.4
2  Wednesday       45      0.5
3   Thursday       23      0.3
4     Friday       51      0.2
5   Saturday       46      0.3
6     Sunday       76      0.6
```

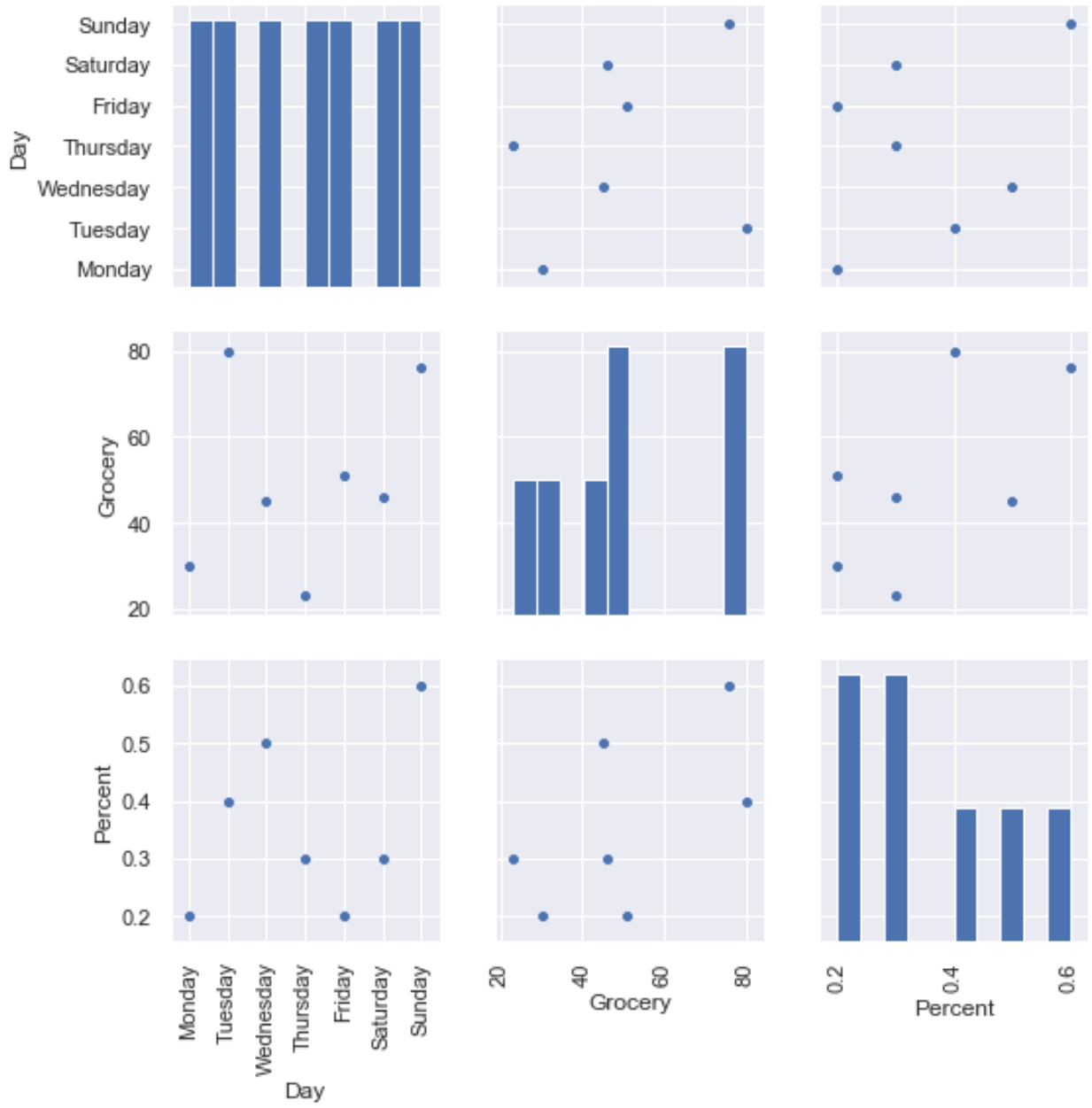
It is quite easy to make a pair plot. When we have string column data we must tell seaborn to include the columns using the string data using the **vars** parameter. The **vars** parameter is used to specify which columns you want to use in your pair plot. If you do not a specify a **vars** parameter the pair plot will plot all columns only having numeric data.

```
# plot pair plot
g=sns.pairplot(data,vars=['Day','Grocery','Percent'])
```

We can specify the have the columns with numeric labels to have the labels rotated vertically rather than horizontally, using the following rotation code.

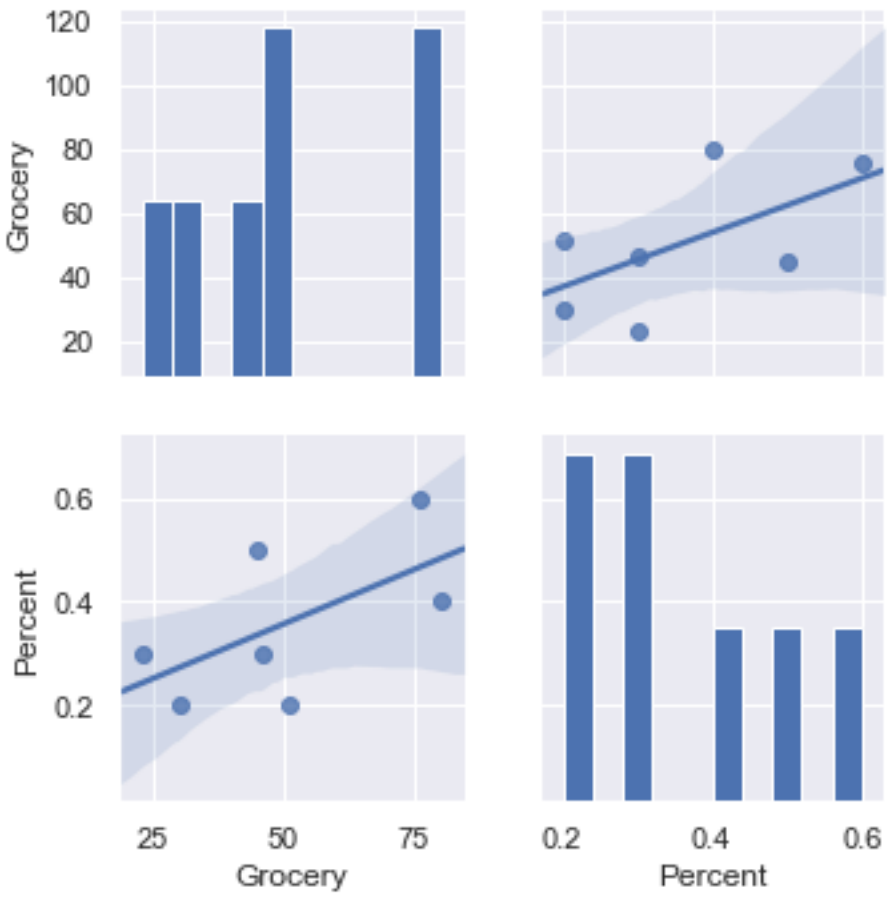
```
g.fig.draw(
   g.fig.canvas.get_renderer()
)
for ax in g.axes.flat:
   ax.set_xticklabels(ax.get_xticklabels(), rotation=90)

plt.show()
```

Our pair plot as follows :



We can also plot the regression line using the **kind** parameter. You cannot plot a regression line for non numeric column data. In this situation we do not need to specify the columns to plot or rotate the x axis labels.

sns.pairplot(data,kind="reg")



Homework

Make a data frame for your monthly expenses for the month. You may have columns for rent, food, entrainment, utilities etc. Plot a distribution plot with a KDE of total weekly expenses. Then make a pair plot. Use labels for the weeks rather than numbers. Week1, week2, week 3 and week4, rotate the labels vertically. Put everything in a python py file called seaborn_homework.py

END